Supporting stepwise, incremental product derivation in product line requirements engineering

Stoiber, R ; Glinz, M

Abstract: Deriving products from a software product line is difficult, particularly when there are many constraints in the variability of the product line. Understanding the impact of variability binding decisions (i.e. of selecting or dismissing features) is a particular challenge: (i) the decisions taken must not violate any variability constraint, and (ii) the effects and consequences of every variability decision need to be understood well. This problem can be reduced significantly with good support both for variability specification and decision making. We have developed an extension of the ADORA language and tool which is capable of modeling and visualizing both the functionality and the variability of a product line in a single model and provides automated reasoning on the variability space. In this paper we describe how our approach supports stepwise, incremental derivation of a product requirements specification from a product line specification. We visualize what has been derived so far, automatically re-evaluate the variability constraints and propagate the results as restrictions on the remaining product derivation options. We demonstrate our approach by showing a sequence of product derivation steps in an example from the industrial automation domain. We claim that our approach both improves the efficiency and quality of the derivation process.

Supporting Stepwise, Incremental Product Derivation in Product Line Requirements Engineering

Reinhard Stoiber, Martin Glinz
Department of Informatics, University of Zurich, Switzerland
Email: {stoiber, glinz}@ifi.uzh.ch

***Abstract*—Deriving products from a software product line is difficult, particularly when there are many constraints in the variability of the product line. Understanding the impact of variability binding decisions (i.e. of selecting or dismissing features) is a particular challenge: (i) the decisions taken must not violate any variability constraint, and (ii) the effects and consequences of every variability decision need to be understood well. This problem can be reduced significantly with good support both for variability specification and decision making. We have developed an extension of the ADORA language and tool which is capable of modeling and visualizing both the functionality and the variability of a product line in a single model and provides automated reasoning on the variability space.**

In this paper we describe how our approach supports stepwise, incremental derivation of a product requirements specification from a product line specification. We visualize what has been derived so far, automatically re-evaluate the variability constraints and propagate the results as restrictions on the remaining product derivation options. We demonstrate our approach by showing a sequence of product derivation steps in an example from the industrial automation domain. We claim that our approach both improves the efficiency and quality of the derivation process.

I. INTRODUCTION

Product derivation is the process of defining a single application product based on a product line variability model. The product derivation process begins with a product line domain variability model, where all variability is unbound and ends with a single concrete product model. During this process, all product line variability needs to be bound, i.e. selected or dismissed.

Product line variability models can become very complex, particularly when the variability is restricted by many constraints. When deriving a product from a product line, an application engineer is challenged with two problems: (i) the decisions he or she takes must not violate the variability constraints, and (ii) he or she needs to understand the effects and consequences of every variability binding decision in order to derive a high-quality product configuration.

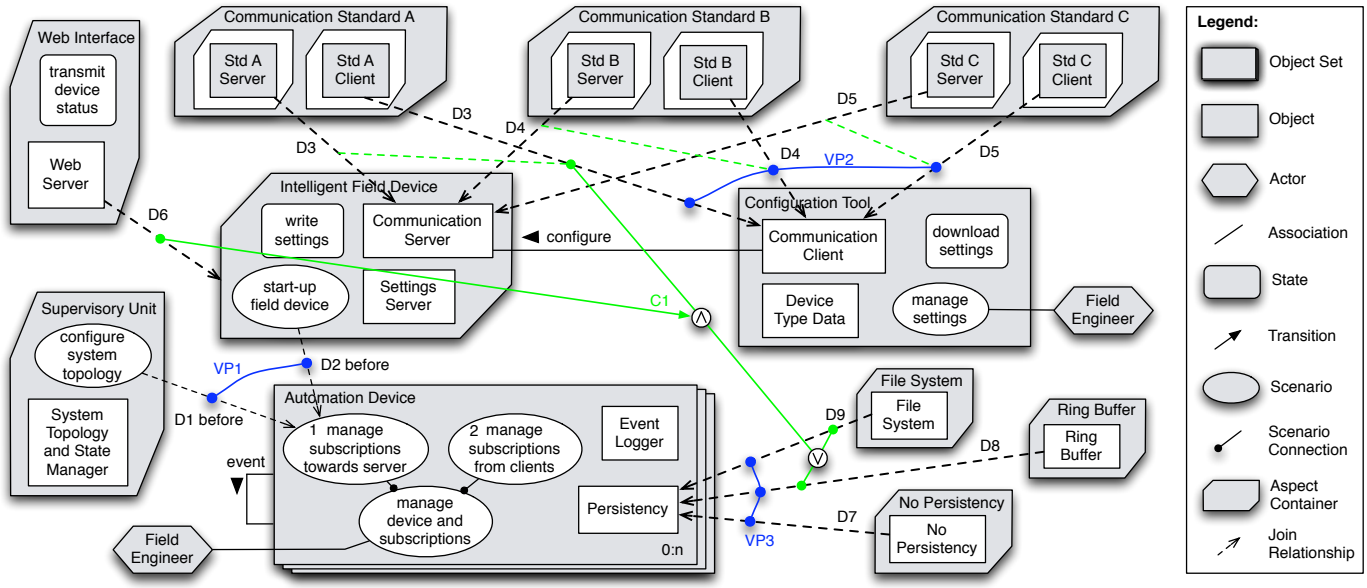
Existing solutions for product derivation from a product line variability model mostly build upon feature-oriented domain analysis [1] or slightly extended versions thereof. However, feature modeling languages have two significant limitations. First, feature models can only define rather simple variability constraints. Schobbens et al. [2] conducted a survey of feature modeling languages that claimed to improve the expressiveness of the original FODA method [1]. According to this survey, current feature modeling languages support only simple types of constraints, mostly based on cardinalities, for example and/or/xor relationships between sub-features with the same parent feature, and *requires*, *excludes* or *mutual exclusion* dependencies between features. For complex constraints, this is too simplistic. For example, a constraint such as $F1\ implies\ F2\ or\ F3$ cannot be specified with a single constraint in a feature model. Second, feature models provide only the names of the available features, but no detailed information about their concrete functionality. In order to comprehend the meaning and impact of a feature, additional documentation or domain expert knowledge is required. Mannion [3], Jarzabek et al. [4] and Czarnecki et al. [5] [6] have addressed this limitation by introducing mappings between features and single textual requirements, requirements documents and UML models, respectively. Thus, with appropriate tool support, the detailed impact of a feature can be shown in other documentation, but this requires a lot of context switching, which is not ideal.

We have developed a new approach to product line requirements modeling that addresses and aims to solve these two problems. Our approach builds on the graphical object-oriented requirements specification language ADORA [7], ADORA's aspect-oriented modeling capabilities [8] which we use for modularizing product line variability, and a new table-based boolean decision modeling concept which we use for managing the product line variability. Using aspects for separately modularizing variability (i.e. variable features) and its composition semantics allows us integrate the variability model into the graphic requirements model and visualize them together.

In this paper, we concentrate on describing our approach to variability decision and constraints modeling and how we can support stepwise, incremental derivation of products. The main idea is to encode the constraints in tables with boolean logic. This allows us to apply existing boolean satisfiability (SAT) solving tools for automated reasoning and verification during product derivation. Using an industrial example, we demonstrate how our techniques enable a stepwise and incremental approach to product derivation.

The remainder of the paper is organized as follows. In Section 2 we introduce an industrial product line requirements specification as a running example and briefly explain the ADORA product line modeling approach. In Section 3 we motivate and describe how we support stepwise and incremental product derivation. In Section 4 we demonstrate the application of our approach to the example introduced

Proceedings of the Fourth International Workshop on Variability Modeling of Software-intensive Systems (VaMoS'10), Linz, Austria, January 2010.



Decision Table

ID	Description / Derivation Question	Design Rationale	Constraints	ifTrue	ifFalse	Decision
D1	Should the automation device be a supervisory unit?	The automation device platform is used for cost and quality reasons.	VP1	¬D2	D2	*undecided*
D2	Should the automation device be an intelligent field device?	The automation device platform is used for cost and quality reasons.	VP1	¬D1	D1 ¬D6	*undecided*
D3	Should the intelligent field device support communic. std. A?	Standard A is the newest and most performant one.	VP2, C1		¬D6	*undecided*
D4	Should the intelligent field device support communic. std. B?	Is a must in some countries.	VP2			*undecided*
D5	Should the intelligent field device support communic. std. C?	C is still required by many legacy systems.	VP2			*undecided*
D6	Should the intelligent field device a web interface?	The web interface allows access over the world wide web.	C1	D3 ¬D7		*undecided*
D7	Is there no memory in the automation device?	Might be interesting for non-critical systems.	VP3	¬D6 ¬D8 ¬D9		*undecided*
D8	Is there a ring buffer in the automation device?	Ring buffers are cheap to implement and fast.	VP3, C1	¬D7 ¬D9		*undecided*
D9	Is there a file system in the automation device?	File system offer huge amounts of storage space.	VP3, C1	¬D7 ¬D8		*undecided*

Variation Points Table

ID	Description / Rationale	minCard	maxCard	Decisions Involved
VP1	An automation device always can be either a supervisory unit or an intelligend field device.	1	1	D1, D2
VP2	There are three different communication standards to configure a field device: standard A, B, and C. At least one must be chosen.	1	3	D3, D4, D5
VP3	There exist three different persistency types: no persistency, a ring buffer, or a file system. One of these must be selected.	1	1	D7, D8, D9

Constraints Table

ID	Description / Rationale	Antecedent	Operator	Consequent
C1	A web interface always requires the communication standard A and a local persistency installed (either ring buffer or FS).	D6	=>	D3 and (D8 or D9)

Fig. 1. An automation device product line specified in the ADORA language.

above. Section 5 discusses scalability issues, related work and concludes.

II. Product Line Domain Modeling: An Industrial Exemplar

To demonstrate our approach, we employ a product line requirements model of industrial automation devices at the product and components level [9]. Fig. 1 shows the graphical requirements specification and the decision and constraints tables of this product line. The graphical notation is briefly explained in the legend on the right-hand side of the figure. The commonality is modeled with abstract objects or object sets and the variable requirements are modularized with aspect containers and graphical join relationships. Every variable join relationship is annotated with a decision item. The details of these decision items are modeled in tables. The subsequent description gives a very brief overview only. For more details, see [9].

A. Overview of the example

In our product line example, which is given in Fig. 1, the commonality of the product line consists of two components: a set of automation devices and a configuration tool. The variability is constituted by the following nine variants: there are two alternatives for the realization of an automation device: a supervisory unit and an intelligent field device. For an intelligent field device, an optional web interface may be added. For the persistency of an automation device, there are three alternatives. Finally, up to three different communication standards for supporting the configuration of an intelligent field device by a configuration tool may be chosen.

In our variability model, every variant is described in detail by the model elements given in its aspect container. The structure of the variability (equivalent to the structure in a feature tree, but only modularizing the *variable* features) is given by join relationships from the variants to the model elements where the variants apply. Every join relationship is annotated with a boolean decision item. These decision items and the constraints applying to them are modeled in tables as

follows.

The *Decision Table* lists all the decisions of the graphic variability model. The columns of the table provide detailed information about decisions, such as a description or constraints. The rightmost column is used to record decisions actually taken in the product derivation process. Initially, in domain modeling, every decision item is undecided by default. Variation points are specified in the *Variation Points Table* with their cardinalities and the decision items involved. Finally, cross-tree constraints that cannot be expressed just as variation points are specified in the *Constraints Table*. Such constraints can be arbitrarily complex formulas in boolean logic. As Fig. 1 shows, ADORA is also capable of visualizing variability constraints graphically in the requirements model [10]. For example, the solid line connecting the join relationship arrows labeled D1 and D2 in Fig. 1 visualizes that these two decision items constitute variation point VP1.

Yet undecided variability is visualized as aspects, according to Fig. 1. As soon as a variability binding decision is taken to true, the corresponding variability is woven into the model at the point(s) designated by the join relationship(s) or, if the decision was taken to false, removed from the model (see Fig. 4).

B. Automated Constraints Analysis

When working with many constraints, the modeler needs support for determining how the constraints impact his or her freedom to take further variability binding decisions. Particularly, he or she needs to know whether a set of decisions is compatible and how taking a certain decision influences the constraints for the remaining, yet undecided decision items. The ADORA tool is capable of providing such support by analyzing the constraints and checking the satisfiability of all currently interesting decision configurations using a boolean satisfiability (SAT) solving tool. The resulting constraint propagations are listed in the columns *ifTrue* and *ifFalse* in the *decision table*. Thus, a modeler can immediately see the consequences of any variability binding decision he or she takes. Moreover, we can guarantee that a partially or fully derived product always satisfies all constraints. This is very similar to Don Batory's idea of building a logic truth maintenance system (LTMS) [11]. In our solution, we additionally calculate constraint propagation previews as follows.

To calculate the constraint propagations, we iterate over all currently undecided decision items in the decision table and decide them once true and once false. For every of these partial configurations, we again iterate over all remaining, still undecided decisions and decide them again once true and once false. All resulting configurations are checked whether or not they satisfy all constraints, using a SAT solver. If a configuration is satisfiable, no propagation is needed. Otherwise, we know that the combination of two decision items violates the constraints. In this case, we record a constraint propagation: if the first decision of the violating configuration was true, we enter the negation of the second decision item into the ifTrue column of the first decision item, or, if the first decision was false, into the ifFalse column. The propagation of constraints is transitive, so for every constraint propagation, we also have to calculate all transitively triggered propagations. This process continues until all potential decisions have been evaluated.

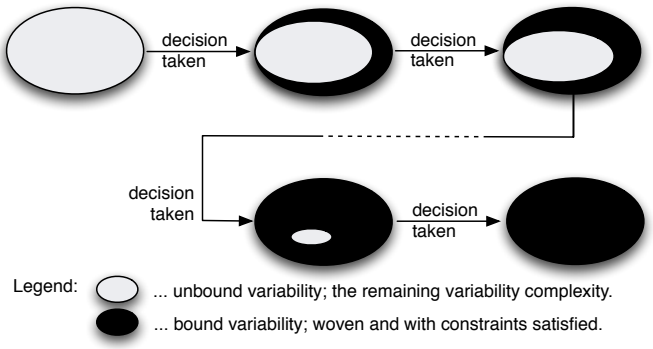


Fig. 2. Stepwise and incremental product derivation in ADORA; with every new variability binding decision the remaining product line variability gets reduced and simpler until a valid final product is derived.

Whenever a decision is actually taken, all constraint propagations are executed. For example, if we decide decision item D2 in the model of Fig. 1 to false, we consequently must decide D1 to true and D6 to false in order to derive a product that satisfies all constraints.

Although SAT solving can be rather computation-intensive, we have not yet experienced any major performance problems. An in-depth performance analysis is subject to future work.

III. Stepwise and Incremental Product Derivation

A. Motivation

A product line domain model contains the full variability and all the variability constraints. Deriving a product configuration (in our case a requirements specification for an individual product) from such a domain model in a single step is almost impossible for any real-size product line: too many options and constraints have to be considered all at once. Hence it is rather straightforward to employ a stepwise process where variability binding decisions are taken one at a time. Fig. 2 illustrates such a process.

However, any mistake in a stepwise product derivation process makes all subsequent steps invalid and, hence, useless. Typical mistakes include decisions that yield an inconsistent configuration or lead into a dead end (i.e. a configuration that can't be further evolved towards the desired product). Consequently, a stepwise product derivation process requires sophisticated tool support, particularly for ensuring consistent intermediate configurations and for analyzing the impact of variability binding decisions, with respect to (i) the effects that the chosen or dismissed variability has on the final product requirements specification, and (ii) the extent to which the current decision restricts the options for the remaining decisions. Furthermore, a tool should also support reverting already taken decisions.

In the subsequent subsection, we describe how we provide such support in ADORA.

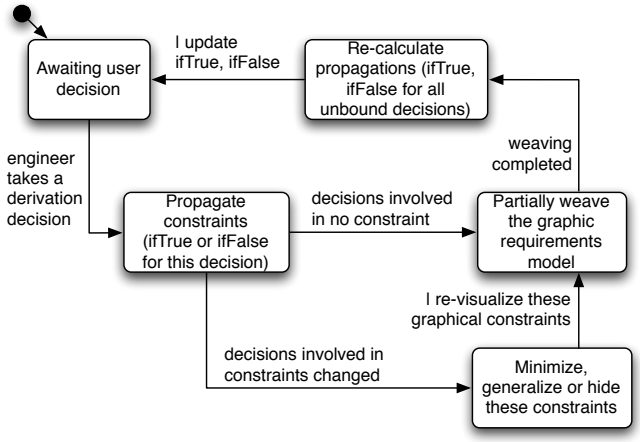


Fig. 3. A statechart describing the behavior of the ADORA tool when executing a variability binding decision taken in a product derivation process.

B. Stepwise and Incremental Product Derivation with ADORA

Product derivation in ADORA is an incremental, step-by-step process. Every time a new variability binding decision is taken or an already taken decision is reverted, the ADORA tool (i) executes the constraint propagations that are associated with this decision, (ii) adapts and re-visualizes all involved graphic variability constraints, (iii) partially weaves the graphic model to reflect the new variability configuration, and (iv) re-calculates the constraint propagations for all still unbound variability decision items (Fig. 3). The derivation process continues until all variability is bound and a concrete product requirements specification has been defined. Note that our current implementation supports forward decision making only. Support for reverting decisions is a part of our ongoing research.

In the remainder of this section, we explain the tasks executed by the ADORA tool in a derivation step (Fig. 3) more in detail. Note that this procedure is fully automatic.

When an engineer takes a decision in the derivation process, the corresponding decision item in the decision table (cf. Fig. 1) is set to true or false. The decision is recorded in the *Decision* column of the decision table. ADORA now executes the tasks described in Fig. 3.

The first task is to propagate the constraints: depending on the truth value of the decision, the decisions listed in the *ifTrue* or in the *ifFalse* columns of the decision table are taken automatically. These decisions transitively propagate their constraints in the same way. All taken decisions are recorded in a decision history. This allows undoing constraint propagations when a decision is undone.

Taking a decision may affect the graphic visualization of variability constraints. For example, if a constraint states that at least one of three options must be selected and a decision is taken that selects one of these options, the constraint is satisfied and no longer needs to be visualized. Therefore, after taking a decision, ADORA needs to *minimize* constraints that are partially satisfied by the decision taken and *hide* those that are fully satisfied. On the other hand, when a taken decision is reverted, the graphic constraint visualizations need to be *generalized* (i.e. restored to their previous state). As the latter is not yet implemented, we focus on the techniques for minimizing and hiding constraints in this paper.

For cardinality based constraints (as listed in the variation points), the minimization of constraints is rather easy and straightforward. For example, if in Fig. 1 one of the decision items D7, D8 or D9 is taken with false, then the variation point VP3 will still be a restriction for the remaining variability configuration space and thus still needs to be displayed for the remaining two decision items. If, on the other hand, one of these three decision items is decided to true, then the other two decision items need to be set to false due to constraint propagations. As a consequence, the VP3 constraint is fully satisfied and will not be visualized any longer.

For the other constraints (as listed in the constraints table), the adaptation is more difficult, since these constraints (i) are defined as implications and (ii) may be arbitrarily complex boolean logic formulas that can span over a large set of decision items. In our example shown in Fig. 1, constraint C1 is such a non-trivial cross-tree constraint. If all involved decision items for such a constraint are undecided and one of these decision items gets bound, then ADORA automatically checks if the constraint is still only partially satisfied, and thus needs to be minimized, or if the constraint is already fully satisfied and thus needs to be hidden from the graphic model. Deciding decision item D6 to false in Fig. 1 is an example for the latter case: the antecedent of constraint C1 becomes false and thus the consequent of this constraint does not need to be enforced anymore. This means that the constraint is satisfied and will be hidden from the graphic model. If we leave D6 undecided and decide D3 to true instead, we have an example for the first case, where constraint C1 needs to be minimized: the logical and operand in the consequent of the constraint is now satisfied and thus this clause needs to be removed from the constraint. This yields the following minimized form of C1: D6 $=>$ D8 or D9.

In the ADORA tool we have solved and implemented the minimization problem with an existing algorithm. Out of several possible algorithms, we chose the Quine McCluskey algorithm [12] that is well known from computer hardware design for simplifying digital circuits. This algorithm is simple to implement and always finds a minimized form of the given constraint. The only disadvantage is that this algorithm computes only one minimized form of the constraint, even when there exist several ones. It could be that a different but equally minimal form of the constraint would be more intuitive to display in the graphical model. This, however, did not turn out as a considerable limitation yet, since it may occur only with very complex constraints.

After the minimizations of the graphical constraints have been computed, the ADORA tool weaves or removes all aspect containers and join relationships associated with the decision taken by the engineer as well as those aspect containers and join relationships associated with the decisions taken due to

constraint propagations. The weaving semantics builds on a slightly extended form of the weaving semantics introduced in [8], which focuses on modularizing cross-cutting concerns with aspects in ADORA. The details of the weaving semantics are beyond the scope of this paper. Examples are provided in the next section, see Fig. 4.

Finally, as Fig. 3 shows, the last task is re-calculating the constraint propagations as described above in subsection II B and updating the *ifTrue* and *ifFalse* columns of the decision table accordingly .

As a final result, ADORA displays a new partially (or fully) woven product line variability model which has less variability than before and is consistent with all constraints. The engineer can now continue with further variability binding decisions in the derivation process.

IV. Example: Semi-Automated Stepwise and Incremental Product Derivation with ADORA

In Fig. 4 we show an incremental, stepwise product derivation in five steps, as implemented in the ADORA tool. As an example, we use the automation devices product line that we introduced in section 2. Fig. 1 presents the fully unbound product line domain model which is the basis for this product derivation process. We assume that a group of engineers wants to derive the requirements specification for a concrete automation device from this product line.

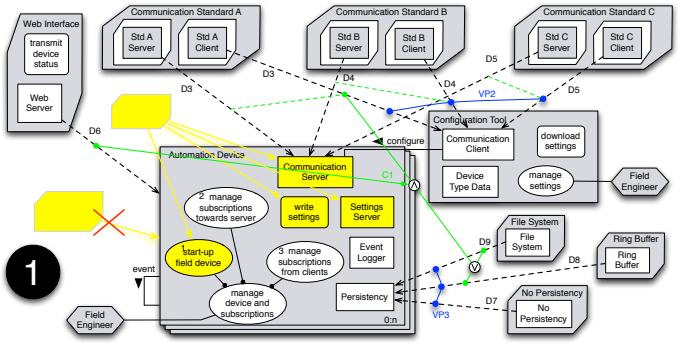
Let's assume that the engineers want the product to be an intelligent field device. As they can see from the product line domain model (cf. Fig. 1), selecting this variant requires to dismiss the variant *Supervisory Unit* (ifTrue column of decision D2 and the graphical constraint VP1). The engineers choose to select this variant and set the decision value of the decision item D2 to true in the ADORA tool. As a consequence, the ADORA tool propagates the constraint in the ifTrue column of decision item D2 and sets decision item D1 to true. Then the tool hides the variation point constraint VP1 because it is now satisfied. Next, the tool performs the weaving operations associated with D2 and D1: The model fragment contained in the *Intelligent Field Device* aspect is woven into the *Automation Device* object set. The *Supervisory Unit* aspect and the join relationship labeled D1 are removed from the model because D1 has been set to false as a constraint propagation. Weaving the *Intelligent Field Device* aspect further requires a redirection of all its incoming join relationships (i.e. the join relationships of the sub-variants of this variant). These join relationships receive new target join points, which are now inside the *Automation Device* object set. Finally, ADORA re-calculates the constraint propagations for all remaining decision items. In this case there are no changes, except that the ifTrue and ifFalse values disappear for decision items D1 and D2 as they have been decided in this step. Diagram 1 in Fig. 4 shows the result of this first product derivation step. All concerned aspects and join relationships are highlighted. Removed items are marked with a cross. Note that these markups are *not* done by the ADORA tool, but have been added manually here for convenience of the reader.

In the second step, the engineers decide to choose the communication standard B to be part of their product and thus set decision item D4 to true. For this decision, no constraint propagations are necessary, because decision item D4 has no values in the ifTrue and ifFalse columns. However, D4 is involved in the variation point constraint VP2. This constraint (minCard 1, maxCard 3 on D3, D4, D5) is satisfied when D4 is decided to true and will be hidden, hence. Next the aspect *Communication Standard B* is woven into the *Automation Device* object set and into the *Configuration Tool* object, according to the join relationships associated with D4 in Diagram 1 in Fig. 4. Finally, the constraint propagations are re-calculated – again there are no changes. Diagram 2 in Fig. 4 shows the resulting intermediate model after step 2.

In step 3 the engineers decide *not* to choose the *Ring Buffer* variant for an implementation of the persistency component and set decision item D8 to false. This decision again does not trigger any constraint propagation. However, it is involved in two variability constraints which both need to be minimized. The variation point constraint VP3 still puts a restriction on the remaining two variants, which now is an alternative. The global constraint C1 is reduced to D6 $=>$ D3 and D9. This means that if the *Web Interface* variant is chosen in a subsequent step, the *Communication Standard A* and the *File System* variants must also be chosen. After these minimizations, the join relationship annotated with D8 is removed from the graphical model and the related aspect is removed as well. Finally, the constraint propagations are re-calculated. This time, the values for the decision items D6, D7 and D9 are modified. D7 and D9 (the *File System* persistency and the *No Persistency* options) are now mutually exclusive alternatives and, as a consequence of constraint C1, D6 (the *Web Interface* variant) definitely requires both D3 (*Communication Standard A*) and D9 (*File System* persistency) to be selected. Conversely, the value in the ifFalse column of decision item D3 tells the engineers that if they would decide not to select *Communication Standard A* in a subsequent step, the *Web Interface* variant could not be chosen anymore.

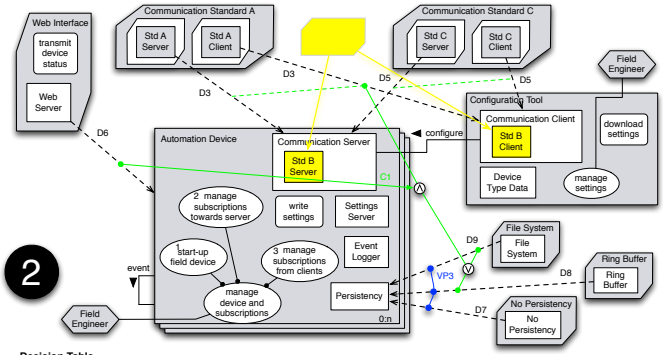
In step 4 the engineers can already reason on the basis of this newer and simplified decision table and graphical model. They decide to choose the *Web Interface* variant to be part of the derived product and set D6 to true. This consequently triggers three constraint propagations: D3 and D9 are set to true because of constraint C1, and D7 is set to false as a transitive consequence of setting D9 to true. All variability constraints are now satisfied and thus hidden from the graphic model. Then the weaving operations are performed and the constraint propagations re-calculated. In the resulting model, only D5 is still undecided.

In step 5 the engineers recognize that only the *Communication Standard C* variant is left as an option in this nearly full configuration. As there are no more constraints, the engineers can freely choose or dismiss this option. Here they decide that they don't need *Communication Standard C* and set the decision item D5 to false. Consequently the tool removes this aspect and the corresponding join relationship from the



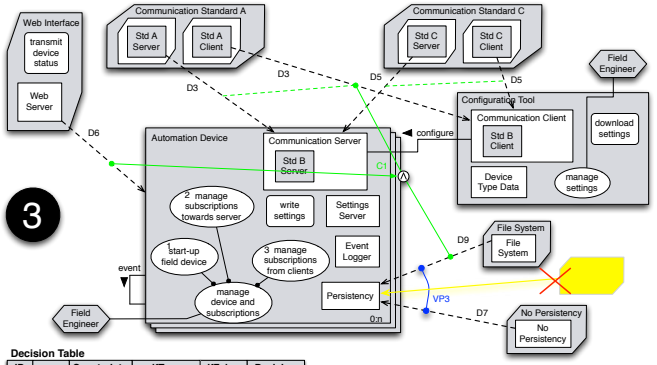
Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1		¬D6	*undecided*
D4		VP2			*undecided*
D5		VP2			*undecided*
D6		C1	D3 ¬D7		*undecided*
D7		VP3	¬D6 ¬D8 ¬D9		*undecided*
D8		VP3, C1	¬D7 ¬D9		*undecided*
D9		VP3, C1	¬D7 ¬D8		*undecided*



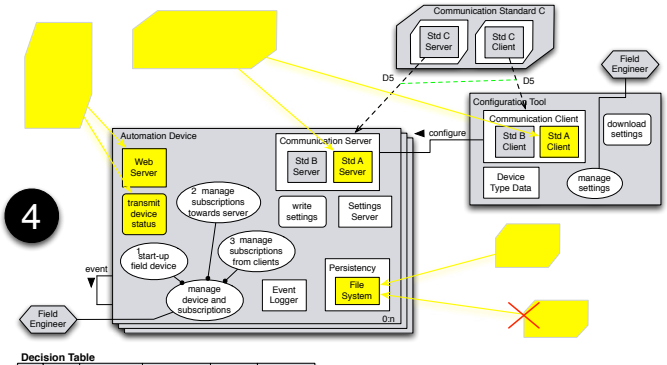
Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1		¬D6	*undecided*
D4		VP2	-	-	***true***
D5		VP2			*undecided*
D6		C1	D3 ¬D7		*undecided*
D7		VP3	¬D6 ¬D8 ¬D9		*undecided*
D8		VP3, C1	¬D7 ¬D9		*undecided*
D9		VP3, C1	¬D7 ¬D8		*undecided*



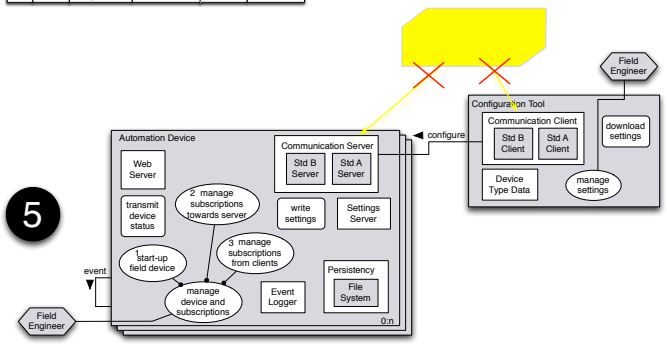
Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1		¬D6	*undecided*
D4		VP2	-	-	***true***
D5		VP2			*undecided*
D6		C1	D3 ¬D7 D9		*undecided*
D7		VP3	¬D6 ¬D9	D9	*undecided*
D8		VP3, C1	-	-	***false***
D9		VP3, C1	¬D7	¬D6 D7	*undecided*



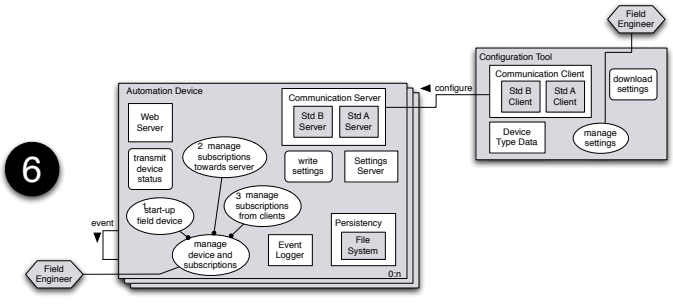
Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1	-	-	***true***
D4		VP2	-	-	***true***
D5		VP2			*undecided*
D6		C1	-	-	***true***
D7		VP3	-	-	***false***
D8		VP3, C1	-	-	***false***
D9		VP3, C1	-	-	***true***



Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1	-	-	***true***
D4		VP2	-	-	***true***
D5		VP2	-	-	***false***
D6		C1	-	-	***true***
D7		VP3	-	-	***false***
D8		VP3, C1	-	-	***false***
D9		VP3, C1	-	-	***true***



Decision Table

ID	...	Constraints	ifTrue	ifFalse	Decision
D1		VP1	-	-	***false***
D2		VP1	-	-	***true***
D3		VP2, C1	-	-	***true***
D4		VP2	-	-	***true***
D5		VP2	-	-	***false***
D6		C1	-	-	***true***
D7		VP3	-	-	***false***
D8		VP3, C1	-	-	***false***
D9		VP3, C1	-	-	***true***

Fig. 4. An exemplary stepwise, incremental product derivation from the product line variability model of Fig. 1. Diagrams 1-5 show how the model changed after the respective derivation step. Diagram 6 shows the final product model. The yellow elements (light grey in B/W prints) and the crosses are only added for illustration purposes.

model. The result is a fully configured product requirements model which is correct (in terms of satisfying all constraints) by construction. Diagram 6 in Fig. 4 shows the final product model.

V. Discussion and Conclusion

A. Scalability

The ADORA language allows modeling the structure, behavior and user interaction of a software system in a single integrated hierarchical diagram. Using only one diagram for modeling a complete system requires sophisticated visualization techniques in order to scale, i.e. to keep large models comprehensible for humans users. In our previous work, we have developed such techniques, for example, fisheye zooming [13] [14], intelligent line-routing [15] and aspect-oriented modeling [8]. In [10] we have argued how these techniques can also be used for product line modeling in ADORA.

Concerning the runtime performance, our current implementation of partial weaving in the ADORA tool is not yet satisfactory for large models, while the performance of our SAT solving implementation doesn't seem to be a problem so far. We plan to do more detailed performance analyses and optimizations both for the our satisfiability solving solution and the aspect weaving in the ADORA tool.

B. Related Work

Stepwise and incremental product derivation is not entirely new. For example, Czarnecki et al. [5] [6] introduced a stepwise derivation process called 'specialization and multi-level configuration of feature models'. In comparison to our approach, this solution has two disadvantages. Firstly, the model elements describing a single feature are scattered over several diagrams (the approach relies on UML). Secondly, feature modeling allows only rather simplistic types of cross-tree constraints, as we argued in the introduction section.

Other authors have addressed the problem of automatic reasoning to leverage the complexity of variability modeling. Thüm et al. [16], for example, recently introduced a grounded framework for reasoning about edits to feature models. They define four particular types of edits that can be made in a feature model. One of them is specialization, which we also use for product derivation. Since they build on the foundations of Batory's work [11], their feature modeling solution allows arbitrarily complex constraints. However, their work so far focuses on feature modeling only. They do not provide any solution to integrate variability modeling with more detailed modeling of the functionality.

White et al. [17] presented a solution that creates mappings between variability models and equivalent constraint satisfaction problems (CSPs) and uses these CSPs for an automated calculation of a sequence of minimal feature adaptations between two different application feature models. As application feature models they typically consider an original feature configuration and a new one that has evolved during the development. White et al. focus on the evolution of an already derived product configuration, while we focus on the product derivation itself in the first place.

Furthermore, there is a range of commercial and open source tools for feature modeling which also support product derivation. However, in terms of feature modeling and automated reasoning, these solutions are less advanced than the one of Thüm et al. [16] and they also have at least the same limitations as Czarnecki et al.'s feature modeling approach [6].

C. Conclusion

We have developed a new variability modeling approach that allows an inherently integrated modeling of features and requirements by building on aspect-oriented modeling, a new table-based boolean decision modeling solution and the ADORA language [7]. Our solution allows the description of arbitrarily complex variability constraints, implements an automated analysis and constraint propagation of these variability constraints and supports incremental and stepwise product derivation. This significantly reduces the complexity and the cognitive load for the model users and improves the understanding of the consequences and the impact of concrete variability binding decisions by human engineers involved in the product derivation process. We claim that already for intermediately complex product lines, such a stepwise and incremental approach to product derivation becomes necessary in order to enable an efficient and intuitive derivation of consistent and valid products.

In our current tool implementation, we have fully implemented the support for stepwise and incremental product derivation that we described in this paper. However, there is still room for improvements and extensions. For example, the generation of human-friendly graphical layouts with our weaving implementation is still a challenge. We did not yet find any really suitable algorithms that always generate satisfying layouts. Further, for our automated reasoning and constraints propagation solution, we do not yet calculate adequate intuitive and minimal sets of constraint propagations needed when already bound decisions are reverted. This is challenging because also the order of the previously taken decisions and their constraint propagations need to be taken into account. The runtime performance of our tool implementation, in particular when weaving models, also needs improvement. Finally, we plan to do empirical evaluations in the near future in order to validate the usefulness of our approach and the support that ADORA provides for real-world product line modeling problems.

Acknowledgment

We would like to thank Ivo Vigan for his programming work and his advice for implementing a SAT solver for ADORA.

References

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[2] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Comput. Netw.*, vol. 51, no. 2, pp. 456–479, 2007.

[3] M. Mannion, "Using first-order logic for product line model validation," in *SPLC'02: Proceedings of the Second Software Product Line Conference.*, ser. Lecture Notes in Computer Science, G. J. Chastek, Ed., vol. 2379. Springer, 2002, pp. 176–187.

[4] S. Jarzabek, W. C. Ong, and H. Zhang, "Handling variant requirements in domain modeling," *J. Syst. Softw.*, vol. 68, no. 3, pp. 171–182, 2003.

[5] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[6] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, "fmp and fmp2rsm: Eclipse plug-ins for modeling features using model templates," in *OOPSLA Companion*, 2005, pp. 200–201.

[7] M. Glinz, S. Berner, and S. Joos, "Object-oriented modeling with ADORA," *Inf. Syst.*, vol. 27, no. 6, pp. 425–444, 2002.

[8] S. Meier, T. Reinhard, R. Stoiber, and M. Glinz, "Modeling and evolving crosscutting concerns in ADORA," in *Aspect-Oriented Requirements Engineering and Architecture Design, 2007. Early Aspects at ICSE: Workshops in*, May 2007.

[9] R. Stoiber and M. Glinz, "Modeling and managing tacit product line requirements knowledge." in *2nd International Workshop on Managing Requirements Engineering Knowledge (MaRK'09)*. IEEE CS, 2009.

[10] R. Stoiber, T. Reinhard, and M. Glinz, "Visualization support for software product line modeling," in *Proceedings of SPLC'08 (Second Volume). 2nd International Workshop on Visualisation in Software Product Line Engineering (ViSPLE'08).*, 2008, pp. 313–322.

[11] D. S. Batory, "Feature models, grammars, and propositional formulas," in *SPLC'05: Proceedings of the 9th International Software Product Line Conference.*, ser. Lecture Notes in Computer Science, J. H. Obbink and K. Pohl, Eds., vol. 3714. Springer, 2005, pp. 7–20.

[12] E. J. McCluskey, "Minimization of boolean functions." *Bell System Technology Journal.*, vol. 35, no. 5, pp. 1417–1444, 1956.

[13] C. Seybold, M. Glinz, S. Meier, and N. Merlo-Schett, "An effective layout adaptation technique for a graphical modeling tool," in *ICSE'03: Proceedings of the 25th International Conference on Software Engineering.*, May 2003, pp. 826–827.

[14] T. Reinhard, S. Meier, R. Stoiber, C. Cramer, and M. Glinz, "Tool support for the navigation in graphical models," in *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 823–826.

[15] T. Reinhard, C. Seybold, S. Meier, M. Glinz, and N. Merlo-Schett, "Human-friendly line routing for hierarchical diagrams," in *ASE '06: Proceedings of the 21st International Conference on Automated Software Engineering.*, Sept. 2006, pp. 273–276.

[16] T. Thüm, D. Batory, and C. Kästner, "Reasoning about edits to feature models," in *ICSE '09: Proceedings of the 31st International Conference on Software Engineering.*, 2009, pp. 254–264.

[17] J. White, D. Benavides, B. Dougherty, and D. C. Schmidt, "Automated reasoning for multi-step software product-line configuration problems." in *SPLC'09: Proceedings of the 13th International Software Product Line Conference.* IEEE CS, 2009.